**6주차 Tetris-1 결과보고서**

전공: 수학과 학년: 4 학번: 20161255 이름: 장원태

1. **실습 시간에 작성한 프로그램의 함수들이 예비보고서에서 작성한, 각 구현 함수들의 pseudo code와 어떻게 달라졌는지 설명하고, 시간 및 공간 복잡도를 보이시오.**

CheckToMove 함수에 대해 예비보고서에서 작성한 pseudo code는 다음과 같다.

|  |
| --- |
| int CheckToMove(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX){  for (i=0;i<BLOCK\_HEIGHT; i++){  for (j=0;j<BLOCK\_WIDTH;j++){  if (block[currentBlock][blockRotate][i][j] == 1) return 0;  else return 1;  }  }  } |

실습시간에 작성한 코드는 다음과 같다.

|  |
| --- |
| int CheckToMove(char f[HEIGHT][WIDTH],int currentBlock,int blockRotate, int blockY, int blockX){  // user code  int i, j;  for (i=0;i<BLOCK\_HEIGHT;i++){  for (j=0;j<BLOCK\_WIDTH;j++){  if (block[currentBlock][blockRotate][i][j] == 1){  if (f[i+blockY][j+blockX] == 1 || i+blockY >= HEIGHT || j+blockX<0 || j+blockX>=WIDTH)  return 0;  }  }  }  return 1;  } |

예비보고서의 경우, 단순히 해당 블록이 차 있으면 0을 리턴하고, 그렇지 않으면 1을 리턴하게끔 코드를 작성했으나, 첫째 문제로 이 알고리즘만으로는 해당 블록이 움직일 수 있는지를 파악할 수 없었고, 단순히 현재 블록의 위치만을 확인하는 것에 그쳤다. 그래서, 블록이 해당 부분으로 움직일 수 있는지를 확인하도록 if문을 추가했다. 이는 블록을 놓으려고 하는 필드에 이미 블록이 있는 경우, 블록의 각 요소의 y좌표가 HEIGHT보다 크거나 같은지(즉, 게임의 틀을 벗어나려고 하는 것이 아닌지), 블록의 각 요소의 x좌표가 0보다 작은지(게임의 왼쪽 틀을 벗어나려고 하는 것이 아닌지), 혹은 WIDTH보다 크거나 같은지(게임의 오른쪽 틀을 벗어나려고 하는 것이 아닌지)를 확인하는 if문이다.

두 번째로, pseudo code에서 썼던 return 1이 해당 if문에서 바로 리턴되는 것이 아니라, 모든 if문을 돌고도 아무 것도 리턴하지 않았다면 그 때 ‘블록이 움직일 수 있다’라고 판단하여 1을 리턴해야 했다. 그래서, return 1을 for문 바깥으로 이동시켰다.

이 함수는 for문에 의해, 시간 복잡도는 O(BLOCK\_HEIGHT \* BLOCK\_WIDTH)가 되고, 공간의 경우 모든 변수가 고정된 크기를 가지므로 공간 복잡도는 O(1)이 된다.

DrawChange 함수의 경우 예비보고서에서 작성한 pseudo code는 다음과 같다.

|  |
| --- |
| void DrawChange(char field[HEIGHT][WIDTH], int command, int currentBlock, int blockRotate, int blockY, int blockX){  switch(command){  case KEY\_UP:  반시계 방향으로 회전;  break;  case KEY\_DOWN:  y--;  break;  case KEY\_LEFT:  x++;  break;  case KEY\_RIGHT:  x--;  break;  }  for (i=0;i<BLOCK\_HEIGHT;i++){  for (j=0;j<BLOCK\_WIDTH;j++){  if (block[currentBlock][prevbR][i][j] == 1){  move(y, x);  printf(“.”);  }  }  }  DrawBlock(blockY, blockX, currentBlock, blockRotate, draw)  } |

이 함수를 실습시간에 구현한 것은 다음과 같다.

|  |
| --- |
| void DrawChange(char f[HEIGHT][WIDTH],int command,int currentBlock,int blockRotate, int blockY, int blockX){  // user code  int i, j;  int tempR = blockRotate; int tempY = blockY; int tempX = blockX;  //1. 이전 블록 정보를 찾는다. ProcessCommand의 switch문을 참조할 것  switch(command){  case KEY\_UP:  tempR = (tempR+3)%4;  break;  case KEY\_DOWN:  tempY--;  break;  case KEY\_LEFT:  tempX++;  break;  case KEY\_RIGHT:  tempX--;  break;  }  //2. 이전 블록 정보를 지운다. DrawBlock함수 참조할 것.  for (i=0; i<4; i++){  for (j=0; j<4; j++){  if (block[currentBlock][tempR][i][j] == 1 && i+tempY>=0){  move(i+tempY+1, j+tempX+1);  printw(".");  }  }  }  //3. 새로운 블록 정보를 그린다.  DrawBlock(blockY, blockX, currentBlock, blockRotate, ' ');  move(HEIGHT, WIDTH+10);  } |

우선 pseudo code에서 사용해야 했던 ‘prevbR’, 즉 입력받은 blockRotate값을 비롯하여 이동하기 전 원래 있던 블록의 x좌표와 y좌표를 저장하기 위해 tempX, tempY, tempR 변수를 선언하여 입력받은 blockX, blockY, blockRotate를 저장하였다. 그 이후에는 pseudo code와 큰 차이 없이, command에 따라 tempX, tempY, tempR값을 조율하고(이전 값을 조율해야 하기 때문에 command의 내용과 반대로 이동시켜야 한다), 이전 블록 정보를 for문을 통해, y좌표와 x좌표를 적절히 이동시켜 지우는 과정을 거쳤다. 그리고, i+tempY >= 0임을 확인하여 해당 조건을 만족하지 않는 이상한 부분을 지우는 일이 없도록 보완했다. 추가로, 실수로 사용한 printf 함수 대신 printw 함수를 사용했다.

이 함수의 경우, 시간 복잡도는 for문만을 고려하면 되므로, O(BLOCK\_HEIGHT \* BLOCK\_WIDTH)가 된다. 공간 복잡도의 경우, 마찬가지로 모든 변수가 고정된 크기를 가지므로 O(1)이 된다.

BlockDown 함수의 경우, 예비보고서에서 작성한 pseudo code는 다음과 같다.

|  |
| --- |
| void BlockDown(int sig){  if (블록을 내릴 수 있음){  blockY++  DrawChange(field, KEY\_DOWN, 다음 블록, blockRotate, blockY, blockX)  }  else{  if (blockY == -1) gameOver=1;  AddBlockToField(field, nextBlock[0], blockRotate, blockY, blockX)  }  } |

이 함수를 실습에서 구현한 것은 다음과 같다.

|  |
| --- |
| void BlockDown(int sig){  // user code  if (CheckToMove(field, nextBlock[0], blockRotate, blockY+1, blockX) == 1){  blockY++;  DrawChange(field, KEY\_DOWN, nextBlock[0], blockRotate, blockY, blockX);  }  else{  if (blockY == -1) gameOver = 1;  AddBlockToField(field, nextBlock[0], blockRotate, blockY, blockX);  score += DeleteLine(field);  PrintScore(score);  nextBlock[0] = nextBlock[1];  nextBlock[1] = rand()%7;  blockRotate = 0; blockY = -1; blockX = (WIDTH/2)-2;    DrawNextBlock(nextBlock);  DrawField();  DrawBlock(blockY, blockX, nextBlock[0], blockRotate, ' ');  }  timed\_out = 0;  //강의자료 p26-27의 플로우차트를 참고한다.  } |

이 함수의 경우, 우선 블록을 내릴 수 있는 경우에 대해서는 pseudo code와 같은 방식으로 진행하였으며, 블록을 내릴 수 있는지는 해당 CheckToMove 함수값이 1인지를 확인함을 통해 파악했다. 블록을 내릴 수 없는 경우, 우선 game over 조건을 추가하는 것과, 해당 블록을 필드값에 추가하는 과정까지는 pseudo code와 같다. 그러나, pseudo code에서는 이 함수에서 점수를 계산하고, 다음 블록을 받으며 초기화 과정까지 취해주어야 한다는 사실을 인지하지 못하여 해당 과정들을 구현하지 않았다. 그래서 블록을 필드값에 추가한 뒤, 완성된 줄이 있으면 지우면서 점수까지 추가 및 출력해주고, 다음 블록을 가져오며, blockRotate값과 block의 x좌표와 y좌표를 모두 초기화하며, 다음 블록을 갱신하고, 필드를 다시 그리고, 이제 나와야 할 블록을 그리는 과정까지 모두 추가했다. 그리고, 블록이 1초에 한번씩 떨어지게 설정하기 위해, 모든 if문이 종료된 후에 timed\_out 변수를 0으로 설정해주는 과정까지 수행했다.

이 함수의 경우 별도의 for문이나 while문도 없으며, 모든 변수가 고정된 크기를 가지므로, 시간 복잡도는 O(1)이고 공간 복잡도도 O(1)이다.

AddBlockToField 함수의 경우, 예비보고서에서 작성한 pseudo code는 다음과 같다.

|  |
| --- |
| void AddBlockToField(char field[HEIGHT][WIDTH], int currentBlock, int blockRotate, int blockY, int blockX){  for (i=0;i<BLOCK\_HEIGHT;i++){  for (j=0;j<BLOCK\_WIDTH;j++){  if (block[currentBlock][blockRotate][i][j] == 1)  field[blockY+i][blockX+j] = 1;  }  }  } |

이 함수를 실습에서 구현한 바는 다음과 같다.

|  |
| --- |
| void AddBlockToField(char f[HEIGHT][WIDTH],int currentBlock,int blockRotate, int blockY, int blockX){  // user code  int i, j;  //Block이 추가된 영역의 필드값을 바꾼다.  for (i=0; i<BLOCK\_HEIGHT; i++){  for (j=0; j<BLOCK\_WIDTH; j++){  if (block[currentBlock][blockRotate][i][j] == 1)  f[blockY+i][blockX+j] = 1;  }  }  } |

이 함수의 경우는 단순히 block의 x좌표와 y좌표를 비롯한 정보를 받아 필드에 그려주기만 하는 함수로, pseudo code와 거의 일치했다. 현재 블록의 x좌표와 y좌표를 바탕으로, 블록이 그려져야 할 위치에 필드값을 추가해준 함수이다.

이 함수의 시간 복잡도의 경우, for문을 고려해야 하므로 O(BLOCK\_HEIGHT \* BLOCK\_WIDTH)가 된다. 공간 복잡도의 경우, 모든 변수가 고정된 크기를 가지므로 O(1)이 된다.

마지막으로 DeleteLine 함수의 경우, 예비보고서에서 작성한 pseudo code는 다음과 같다.

|  |
| --- |
| Int DeleteLine(char field[HEIGHT][WIDTH]){  flag = -1;  for (i=1;i<HEIGHT;i++){  for (j=0;j<WIDTH;j++){  if (field[i][j] == 0){flag=0; break;}  }  }  if (flag == 1){  온전히 채워진 라인 삭제;  점수 계산;  return 점수;  }  } |

이를 실습에서 구현한 바는 다음과 같다.

|  |
| --- |
| int DeleteLine(char f[HEIGHT][WIDTH]){  // user code  int i, j, k;  int check = 1;  int deletedline = 0;  //1. 필드를 탐색하여, 꽉 찬 구간이 있는지 탐색한다.  for (i=1; i<HEIGHT; i++){  check = 1;  for (j=0; j<WIDTH; j++){  if (f[i][j] == 0){  check = 0;  break;  }  }  //2. 꽉 찬 구간이 있으면 해당 구간을 지운다. 즉, 해당 구간으로 필드값을 한칸씩 내린다.  if (check){  for (j=i; j>0; j--){  for (k=0; k<WIDTH; k++)  f[j][k] = f[j-1][k];  }  deletedline++;  }  }  return deletedline\*deletedline\*100;  } |

우선, pseudo code의 경우 각 for문의 범위를 잘못 잡아, flag가 제 구실을 하지 못하게 되었다. 예를 들어, 검색 과정에서 앞선 줄이 차있지 않음을 확인한 경우, 그 다음 줄이 차있음에도 불구하고 flag가 0으로 설정이 되어있어 온전히 채워진 라인을 삭제하지 못하는 일이 발생했다. 그래서, 바깥 for문(i에 대한 for문)이 해당 구간을 지우는 부분(점수를 리턴하기 전)까지 모두 포함할 수 있게 수정했다. 그리고, 플래그 역할을 하는 check 변수가 다음 줄을 검색하기 전에 다시 초기화되게끔(1로 초기화) 하여 flag가 제 구실을 할 수 있도록 수정했다.

그리고, flag를 -1로 설정해놓고, 채워진 라인이 있는 경우를 확인할 때 flag가 1인지를 확인하게 했던 실수가 있어 이를 다시 고쳐주었다. 실습 시간에는 해당 라인이 꽉 차 있지 않다면 check을 0으로 바꾸고, 꽉 차 있다면 check이 1인 상태로 남게 되며, check이 0이 아니면(이 경우에는 1이면) 온전히 채워진 라인을 삭제하고 점수를 그만큼 더하게 설정했다. 이 때, 단순히 삭제만 하면 위에 있는 블록이 자동으로 아래로 내려오지 않으므로, for문을 이용하여 위에 있는 블록을 모두 아래로 내리는 과정까지 추가했다. 추가적으로, 점수를 리턴하는 것은 모든 과정이 끝나고 이루어져야 하므로, 점수 리턴은 바깥 for문까지 모두 끝난 뒤에 이루어지도록 위치를 바꿨다.

이 함수의 시간 복잡도는, 기본적으로 블록의 HEIGHT과 WIDTH만큼 for문이 진행되며, 삭제된 라인이 있다면 그 라인 수만큼 또 for문이 진행된다. 따라서, 시간 복잡도는 O(BLOCK\_HEIGHT \* BLOCK\_WIDTH \* deletedline)이 된다. 공간 복잡도의 경우, 모든 변수가 고정된 크기를 가지므로 O(1)이 된다.

1. **테트리스 프로젝트 1주차 숙제 문제를 해결하기 위한 pseudo code를 기술하고, 시간 및 공간 복잡도를 보이시오.**
2. 그림자 기능

DrawBlock 함수의 경우는 내용 자체는 변화가 없으나, 기존에 DrawBlock 함수를 호출했던 부분을 모두 DrawBlockWithFeatures 함수로 대체하도록 하고, DrawBlockWithFeatures 함수 내에서 DrawBlock 함수가 호출되는 형식으로 바꾸면 된다. DrawBlock 함수 자체는 달라지는 바가 없다.

DrawShadow 함수의 경우 다음과 같이 작성할 수 있다.

|  |
| --- |
| Void DrawShadow(int y, int x, int blockID, int blockRotate, char tile){  While (블록을 내릴 수 있음){  y++;  }  DrawBlock(y, x, blockID, blockRotate, ‘/’);  } |

현재 블록에서 y좌표만 더해주며 그것이 더 이상 내려갈 수 없을 때까지 쭉 내려가, 그 위치에 같은 블록을 ‘/’ 타일로 그려주면 된다.

이 함수의 시간 복잡도는 내려갈 수 있는 y좌표에 비례하므로 O(1)이 되고, 모든 변수가 고정된 크기를 가지므로 공간 복잡도도 O(1)이 된다.

DrawBlockWithFeatures 함수의 경우 다음과 같이 작성할 수 있다.

|  |
| --- |
| Void DrawBlockWithFeatures(int y, int x, int blockID, int blockRotate){  DrawBlock(y, x, blockID, blockRotate, “ “);  DrawShadow(y, x, blockID, blockRotate);  } |

이는 단순히 두 개의 함수를 호출하는 과정으로, 시간 복잡도와 공간 복잡도 모두 O(1)이 된다.

이 때, 블록의 그림자 역시 DrawChange 함수에 의해 영향을 받아야 하므로, DrawChange 함수에서 해당 그림자 부분을 지우는 부분을 추가해야 한다. 이는 while문으로 해당 그림자의 위치를 찾고, 기존에 있던 for문에서 해당 부분을 지우는 과정을 추가하면 된다. 이는 시간 복잡도와 공간 복잡도에 특별히 영향을 미치지 않는다.

1. 2개의 블록 미리 보여주기

다른 코드를 수정하기 앞서, 블록의 개수는 이제 3개가 되므로 tetris.h 파일에서 블록의 개수를 2개에서 3개로 수정해야 한다.

InitTetris 함수의 경우 다음과 같이 수정할 수 있다.

|  |
| --- |
| void InitTetris(){  int i,j;  for(j=0;j<HEIGHT;j++)  for(i=0;i<WIDTH;i++)  field[j][i]=0;  nextBlock[0]=rand()%7;  nextBlock[1]=rand()%7;  nextBlock[2]=rand()%7; // 블록 하나 더 추가  blockRotate=0;  blockY=-1;  blockX=WIDTH/2-2;  score=0;  gameOver=0;  timed\_out=0;  DrawOutline();  DrawField();  DrawBlock(blockY,blockX,nextBlock[0],blockRotate,' ');  DrawNextBlock(nextBlock);  PrintScore(score);  } |

시간 복잡도는 for문만 고려하면 되므로 O(HEIGHT\*WIDTH)이고, 필드에 의해 공간 복잡도도 O(HEIGHT\*WIDTH)가 된다. 이는 수정 전 함수와 같다.

DrawNextBlock 함수의 경우 다음과 같이 수정할 수 있다.

|  |
| --- |
| void DrawNextBlock(int \*nextBlock){  int i, j;  for( i = 0; i < 4; i++ ){  move(4+i,WIDTH+13);  for( j = 0; j < 4; j++ ){  if( block[nextBlock[1]][0][i][j] == 1 ){  attron(A\_REVERSE);  printw(" ");  attroff(A\_REVERSE);  }  else printw(" ");  }  }  // 세 번째 블록 그리는 과정 추가  For (i=0; i<4; i++){  커서를 두 번째 블록 칸 아래로 이동;  For (j=0; j<4; j++){  If (block[nextBlock[2]][0][i][j] == 1){  Attron(A\_REVERSE);  Printw(“ “);  Attroff(A\_REVERSE);  }  }  }  } |

시간 복잡도의 경우, 기존에 하던 작업을 2번 하는 것뿐 기존과 같이 O(BLOCK\_HEIGHT \* BLOCK\_WIDTH)가 된다. 공간 복잡도의 경우 모든 변수가 고정된 크기를 가지므로 O(1)이다.

추가적으로, 세 번째 추가 블록을 그려줌에 따라 DrawOutline 함수에서 세 번째 블록을 보여주는 공간의 테두리를 그리고, score를 보여주는 공간의 테두리를 그만큼 내려줘야 한다. 이에 따른 시간 복잡도는 O(1), 공간 복잡도는 O(1)이 되며 달라지지 않는다.

BlockDown 함수의 경우 다음과 같이 수정할 수 있다.

|  |
| --- |
| void BlockDown(int sig){  // user code  if (CheckToMove(field, nextBlock[0], blockRotate, blockY+1, blockX) == 1){  blockY++;  DrawChange(field, KEY\_DOWN, nextBlock[0], blockRotate, blockY, blockX);  }  else{  if (blockY == -1) gameOver = 1;  AddBlockToField(field, nextBlock[0], blockRotate, blockY, blockX);  score += DeleteLine(field);  PrintScore(score);  nextBlock[0] = nextBlock[1];  nextBlock[1] = nextBlock[2] // 변경할 부분  nextBlock[2] = rand()%7; // 추가할 부분  blockRotate = 0; blockY = -1; blockX = (WIDTH/2)-2;    DrawNextBlock(nextBlock);  DrawField();  DrawBlock(blockY, blockX, nextBlock[0], blockRotate, ' ');  }  timed\_out = 0;  //강의자료 p26-27의 플로우차트를 참고한다.  } |

다음 블록을 가져오는 과정, 즉 세 번째 블록을 가져오는 과정을 추가함에 따라 적절하게 수정해주어야 한다. 시간 복잡도와 공간 복잡도 모두 O(1)로 이전과 동일하다.

1. 닿은 면적만큼 score 증가하기

AddBlockToField 함수의 경우, 다음과 같이 수정할 수 있다.

|  |
| --- |
| int AddBlockToField(char f[HEIGHT][WIDTH],int currentBlock,int blockRotate, int blockY, int blockX){  // user code  int i, j, touched; // touched 변수 추가  //Block이 추가된 영역의 필드값을 바꾼다.  for (i=0; i<BLOCK\_HEIGHT; i++){  for (j=0; j<BLOCK\_WIDTH; j++){  if (block[currentBlock][blockRotate][i][j] == 1){  f[blockY+i][blockX+j] = 1;  if (바로 아래에 블록이 차있거나, 맨 밑줄이면) touched++; // 추가할 부분  }  }  }  Return touched\*10  } |

이 함수의 경우 기존과 다르게 점수를 리턴해야 하므로 int형으로 변환하며, 이는 tetris.h 파일에도 반영되어야 한다. 그리고, 닿은 면적만큼 점수가 비례하여 증가하도록, touched 변수를 선언하여 아래와 닿은 만큼 increment해준다. 그리고, 그의 10배만큼을 리턴한다.

시간 복잡도의 경우, 기존 for문 안에 if문이 생기고 touched 변수가 생긴 것 외에 없으므로 O(BLOCK\_HEIGHT \* BLOCK\_WIDTH)로 같고, 공간 복잡도는 O(1)이다.

BlockDown 함수의 경우 다음과 같이 수정할 수 있다. (위의 b 항목에 있던 내용과는 별개로 작성한다.)

|  |
| --- |
| void BlockDown(int sig){  // user code  if (CheckToMove(field, nextBlock[0], blockRotate, blockY+1, blockX) == 1){  blockY++;  DrawChange(field, KEY\_DOWN, nextBlock[0], blockRotate, blockY, blockX);  }  else{  if (blockY == -1) gameOver = 1;  score += AddBlockToField(field, nextBlock[0], blockRotate, blockY, blockX); // 변경할 부분  score += DeleteLine(field);  PrintScore(score);  nextBlock[0] = nextBlock[1];  nextBlock[1] = rand()%7;  blockRotate = 0; blockY = -1; blockX = (WIDTH/2)-2;    DrawNextBlock(nextBlock);  DrawField();  DrawBlock(blockY, blockX, nextBlock[0], blockRotate, ' ');  }  timed\_out = 0;  //강의자료 p26-27의 플로우차트를 참고한다.  } |

위의 AddBlockToField 함수가 점수를 리턴하게 변경되었으므로, 이에 따라 BlockDown 함수에서 그만큼 점수가 추가되게 바꾼다. 이 외에는 변경점이 없으므로, 시간 복잡도와 공간 복잡도 모두 O(1)로 기존과 동일하다.